Universidade Federal de Uberlândia

FACOM – Bacharelado em Ciência da Computação

GBC052 – Análise de Algoritmos

Professor: Paulo Henrique Ribeiro Gabriel

Atividade Prática da disciplina de Análise de Algoritmos

Felipe Costa Mendes - 11421BCC005

Thiago Lucas Carvalho - 11521BCC012

Uberlândia – 2020

1. Introdução

O presente trabalho foi desenvolvido em conjunto pelos discentes Felipe Costa Mendes e Thiago Lucas Carvalho para a disciplina de Análise de Algoritmos, ministrada pelo professor Paulo Henrique Ribeiro Gabriel. Seu objetivo principal é comparar a complexidade de dois algoritmos de multiplicação de matrizes.

Este relatório visa documentar todo o processo realizado pelo grupo para cumprir às exigências estabelecidas pelo professor nas diretrizes do projeto. Ele está organizado em cinco partes principais. A primeira parte é a introdução, que visa explicar os objetivos deste trabalho e a organização do mesmo. A segunda parte é a descrição do problema, onde serão detalhadas as diretrizes do projeto. Na terceira parte encontra-se alguns aspectos da implementação do código fonte criado. Na quarta parte encontram-se os experimento e resultados obtidos. Na quinta e ultima parte estão as considerações finais acerca do trabalho.

2. Descrição do Problema

A operação de multiplicação de matrizes é, ao lado da soma, uma das operações mais fundamentais sobre a manipulação de matrizes. Sejam duas matrizes ***X*** e ***Y***, com dimensões *n* x *m* e *m* x *p*, respectivamente. Por teoria, esta operação é possível somente se quantidade de colunas de ***X*** é igual à quantidade de linha de ***Y***, resultando assim em uma matriz ***Z*** resultante com dimensões *n* x *p*.

Podemos definir os passos necessários para esta operação no algoritmo da Figura 1. Ele recebe como parâmetros duas matrizes que respeitam a teoria descrita anteriormente e retorna uma matriz resultante.

A complexidade deste algoritmo pode ser facilmente estimada supondo que ambas as matrizes são quadradas, ou seja, possuem a mesma quantidade de linhas e colunas. Dessa forma, encontramos uma complexidade polinomial de grau três, ou seja, a linha 6 do algoritmo será executada $\Theta(n^3)$ vezes.

Algoritmo 1: Multiplicação de duas matrizes.

```
   Entrada: Matrizes X e Y
   Saída: Produto das matrizes X e Y
1  Seja Z uma matriz de dimensão n × p;
2  para i ← 1 até n faça
3  |  para j ← 1 até p faça
4  |  |  Z_ij ← 0;
5  |  |  para k ← 1 até m faça
6  |  |  |  Z_ij ← Z_ij + X_ik · Y_kj;
7  |  |  fim
8  |  fim
9  fim
10 retorna Z
```

Figura 1: Algoritmo simples de multiplicação de matrizes.

Uma alternativa mais eficiente na multiplicação de matrizes quadradas é o Algoritmo de Strassen, proposto em 1969. Este algoritmo se baseia na propriedade de divisão e conquista e visa dividir as matrizes *X* e *Y* em quatro blocos, cada um com dimensão *n/2* x *n/2*. Desse modo, a matriz resultante XY seria representada conforme mostrado na Figura 2, onde cada bloco seria tratado como um elemento da matriz.

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix},\ Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix} \qquad XY = \begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$$

Figura 2: Representação da divisão da matriz em blocos.

Contudo, a grande melhoria do algoritmo de Strassen em relação ao algoritmo da Figura 1 foi a redução da quantidade de produtos matriciais de oito para sete. Dessa forma, a matriz resultante *XY* não seria mais obtida da maneira descrita na Figura 2, e sim seguindo as regras mostradas na Figura 3. Com isso, foi possível obter um algoritmo de multiplicação de matrizes de complexidade O ($n^{\log_2 7}$).

$$XY = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix},$$

sendo que:

- $P_1 = A(F - H)$
- $P_2 = (A + B)H$
- $P_3 = (C + D)E$
- $P_4 = D(G - E)$
- $P_5 = (A + D)(E + H)$
- $P_6 = (B - D)(G + H)$
- $P_7 = (A - C)(E + F)$

Figura 3: Operações do Algoritmo de Strassen.

Nosso objetivo neste projeto é, então, implementar os algoritmos de multiplicação simples de matrizes e o algoritmo de Strassen. Feito isso, podemos testá-los e demonstrar a eficiência do Algoritmo de Strassen sob o algoritmo da Figura 1. Esta análise será realizada comparando o tempo de execução de ambos os algoritmos.

3. Implementação

Iniciamos este trabalho definindo qual linguagem de programação seria utilizada nesta atividade. Optamos pela linguagem C devido à facilidade de se manipular

matrizes e também pela simplicidade do projeto, permitindo que menos fatores comprometesse a medição do tempo de execução do programa.

As matrizes utilizadas neste trabalho serão quadradas e de ordem 2^n para facilitar os cálculos, visto o algoritmo de Strassen divide a matriz em blocos com dimensões *n/2* x *n/2*. Além disso, elas serão alocadas dinamicamente na memória utilizando a técnica da alocação única. Isso significa que os elementos da matriz serão alocados em um único vetor linear, podendo ser acessados por meio da aritmética de ponteiros.

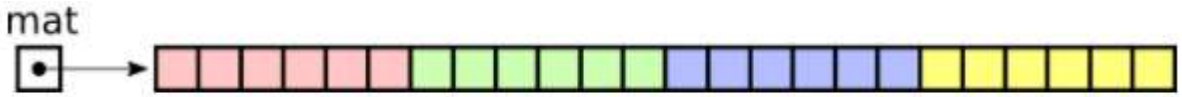

Figura 4: Representação da alocação única de matriz.

Com base nessa estrutura e seguindo a lógica do algoritmo da Figura 1, construímos uma função equivalente em linguagem C. Esta função recebe como parâmetros duas matrizes passadas por referência, ou seja, dois ponteiros indicando a posição de memória do primeiro elemento das matrizes ***m1*** e ***m2***. Isto é feito para evitar a cópia desnecessária de valores, uma vez que as matrizes já foram alocadas na memória. Ao final, a função retorna a matriz resultante ***z***.

```c
int multiplicaMatrizes(int *m1, int *m2)
{
    int *z = criaMatriz(z);

    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < N; j++)
        {
            z[(i * N) + j] = 0;
            for(int k = 0; k < N; k++)
            {
                z[(i * N) + j] = z[(i * N) + j] + m1[(i * N) + k] * m2[(k * N) + j];
            }
        }
    }

    return z;
}
```

Figura 5: Algoritmo simples de multiplicação de matrizes em C.

Seguindo as regras mostradas na Figura 2 e Figura 3, implementamos o algoritmo de Strassen. Ele recebe como parâmetros dois ponteiros para as matrizes de entrada ***A*** e ***B***, um ponteiro para a matriz resultante ***C*** e dois valores ***m*** e ***n***. Na prática, esses dois últimos parâmetros terão o mesmo valor igual à ordem das matrizes. Eles serão utilizados na aritmética de ponteiros para percorrermos as matrizes.

```
int MultiplicaStrassen(int *A, int *B, int *C, int m, int n)
{
    if(m == 2)
    {
        //p1 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1])
        int p1 = (*A + *(A + n + 1)) * (*B + *(B + n + 1));

        //p2 = (A[1][0]+A[1][1])*B[0][0]
        int p2 = (*(A + n) + *(A + n + 1)) * (*B);

        //p3 = A[0][0] * (B[0][1] - B[1][1])
        int p3 = (*A) * (*(B + 1) - *(B + n + 1));

        //p4 = A[1][1] * (B[1][0] - B[0][0])
        int p4 = (*(A + n + 1)) * (*(B + n) - *B);

        //p5 = (A[0][0] + A[0][1]) * B[1][1]
        int p5 = (*A + *(A + 1)) * (*(B + n + 1));

        //p6 = (A[1][0] - A[0][0]) * (B[0][0] + B[0][1])
        int p6 = (*(A + n) - *A) * (*B + *(B + 1));

        //p7 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);
        int p7 = (*(A + 1)-*(A + n + 1)) * (*(B + n) + *(B + n + 1));

        *C       = *C + p1 + p4 - p5 + p7;          //C[0][0]
        *(C+1)   = *(C + 1) + p3 + p5;              //C[0][1]
        *(C+n)   = *(C + n) + p2 + p4;              //C[1][0]
        *(C+n+1) = *(C + n + 1) + p1 + p3 - p2 + p6;  //C[1][1]
    }
```

Figura 6: Caso base do Algoritmo de Strassen.

```
    else
    {
        m = m/2;
        MultiplicaStrassen(A          ,B          ,C          ,m,n);
        MultiplicaStrassen(A          ,B+m        ,C+m        ,m,n);
        MultiplicaStrassen(A+m        ,B+m*n      ,C          ,m,n);
        MultiplicaStrassen(A+m        ,B+m*(n+1)  ,C+m        ,m,n);
        MultiplicaStrassen(A+m*n      ,B          ,C+m*n      ,m,n);
        MultiplicaStrassen(A+m*n      ,B+m        ,C+m*(n+1)  ,m,n);
        MultiplicaStrassen(A+m*(n+1)  ,B+m*n      ,C+m*n      ,m,n);
        MultiplicaStrassen(A+m*(n+1)  ,B+m*(n+1)  ,C+m*(n+1)  ,m,n);
    }
}
```

Figura 7: Chamada recursiva do Algoritmo de Strassen.

Implementadas as duas funções principais do trabalho, criamos as demais funções auxiliares responsáveis por alocar memória da matriz, inicializá-las com valores aleatórios ou com valor zero e imprimir o resultado obtido em tela. Por fim, adicionamos as linhas de código responsável por calcular o tempo gasto de cada algoritmo, que considera apenas a execução das funções descritas acima.

Na imagem abaixo, é possível ver o resultado para a execução do programa com N = 4, ou seja, com matrizes de ordem 4. Vale ressaltar que os tempos de execução mostrados na imagem estão zerados, pois como as matrizes são de ordem baixa, o tempo de cada um dos algoritmos é substancialmente pequeno.

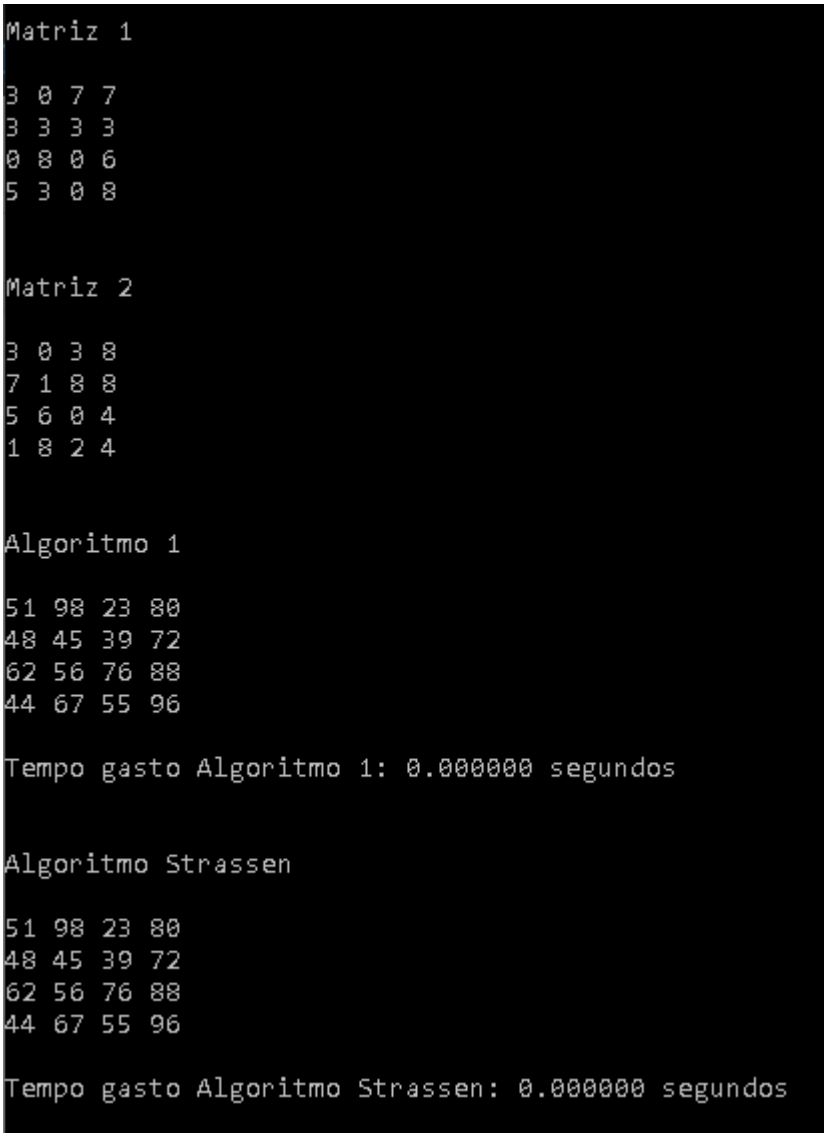

Figura 8: Execução do programa para matrizes de ordem 4.

4. Experimentos e Resultados

Finalizada a implementação dos algoritmos de multiplicação de matrizes, seguimos para os testes com diferentes valores de N. Para isso, comentamos as

linhas de código responsáveis por imprimir as matrizes em tela. Dessa forma, podemos focar apenas na análise do tempo de execução.

Foram criadas 30 réplicas do experimento, cada uma com apenas um par de matrizes de entrada. Os valores de N respeitam a potência de 2^n e os valores das matrizes são gerados aleatoriamente dentro de intervalo de 0 a 10. Podemos ver os resultados de um dos experimentos na tabela e gráfico mostrados abaixo.

Valor de N	Tempo Algoritmo 1	Tempo Algoritmo de Strassen
2	0	0
4	0	0
8	0	0
16	0	0
32	0	0
64	0,002	0,002
128	0,014	0,008
256	0,134	0,056
512	1	0,278
1024	11,164	2,349
2048	165,467	18,407

Tabela 1: Comparação de tempo entre o algoritmo 1 e o algoritmo de Strassen.

Figura 9: Crescimento do tempo de execução das funções.

Para realizar a execução do programa 30 vezes, alteramos o código fonte para calcular a soma total de tempo, a média e o desvio padrão de cada um dos algoritmos. O resultado pode ser visto na figura abaixo.

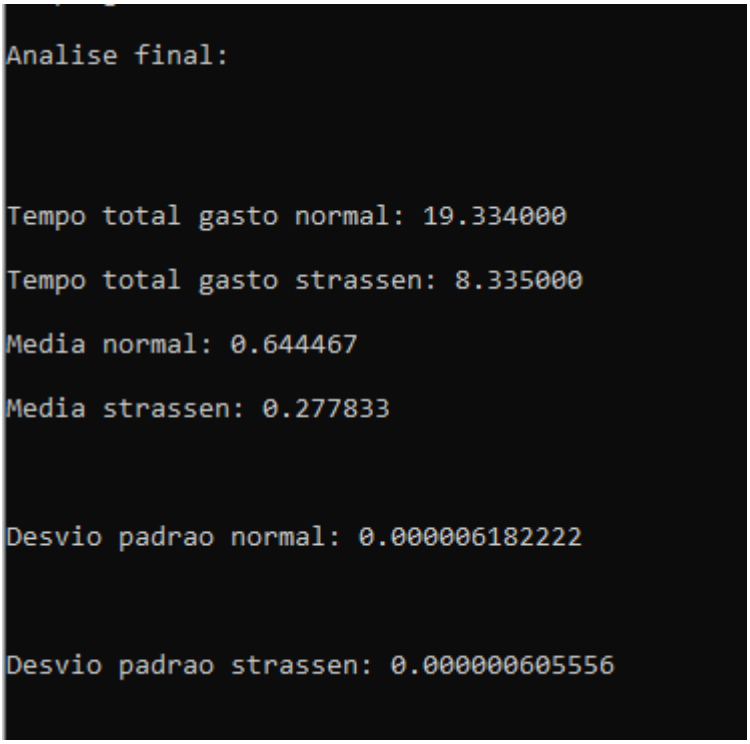

Figura 10: Soma, media e desvio padrão.

5. Considerações finais

Com base nos experimentos realizados e nos resultados obtidos, ficou clara a eficiência do algoritmo de Strassen sob um algoritmo simples de multiplicação de matrizes. Durantes os testes realizados, para N maiores que 4096, por exemplo, o algoritmo comum se torna inviável, demorando cerca de 30 minutos para finalizar a execução. Isso mostra como a técnica de divisão e conquista aliada à análise de algoritmo são ferramentas essenciais para a otimização de diversos algoritmos computacionais.